Docket No. 50277-2235 *Patent*

UNITED STATES PATENT APPLICATION

FOR

DIRECT LOADING OF SEMISTRUCTURED DATA

INVENTORS:

NAMIT JAIN
NIPUN AGARWAL
RAVI MURTHY

PREPARED BY:

HICKMAN PALERMO TRUONG & BECKER LLP
1600 WILLOW STREET
SAN JOSE, CALIFORNIA 95125
(408) 414-1080

ASSIGNEE:

ORACLE INTERNATIONAL CORPORATION
500 ORACLE PARKWAY
REDWOOD SHORES, CA 94065

"Express Mail" mailing label number EV322192481US

Date of Deposit August 25, 2003

DIRECT LOADING OF SEMISTRUCTURED DATA

RELATED APPLICATIONS

[0001] The present application is related to the following U.S. Patent Applications, the entire contents of which are incorporated herein by reference for all purposes:

[0002] U.S. Patent Application Serial No. 10/192,411, filed on July 9, 2002, entitled OPAQUE TYPES, by Rajagopalan Govindarajan, Viswanathan Krishnamurthy, and Anil Nori;

[0003] U.S. Patent Application Serial No. 10/260,138, filed on September 27, 2002, entitled OPERATORS FOR ACCESSING HIERARCHICAL DATA IN A RELATIONAL SYSTEM, by Nipun Agarwal, Ravi Murthy, Eric Sedlar, Sivasankaran Chandrasekar and Fei Ge;

[0004] U.S. Patent Application Serial No. 10/260,384, filed on September 27, 2002, entitled PROVIDING A CONSISTENT HIERARCHICAL ABSTRACTION OF RELATIONAL DATA, by Nipun Agarwal, Eric Sedlar, Ravi Murthy and Namit Jain;

[0005] U.S. Patent Application Serial No. 10/259,278, filed on September 27, 2002, entitled MECHANISM FOR MAPPING XML SCHEMAS TO OBJECT-RELATIONAL DATABASE SYSTEMS, by Ravi Murthy, Muralidhar Krishnaprasad, Sivasankaran Chandrasekar, Eric Sedlar, Vishu Krishnamurthy and Nipun Agarwal;

[0006] U.S. Patent Application Serial No. 10/260,161, filed on September 27, 2002, entitled INDEXING TO EFFICIENTLY MANAGE VERSIONED DATA IN A DATABASE SYSTEM, by Nipun Agarwal, Eric Sedlar and Ravi Murthy;

[0007] U.S. Patent Application Serial No. 10/256,524, filed on September 27, 2002, entitled MECHANISMS FOR STORING CONTENT AND PROPERTIES OF

HIERARCHICALLY ORGANIZED RESOURCES, by Ravi Murthy, Eric Sedlar, Nipun Agarwal, and Neema Jalali;

[0008] U.S. Patent Application Serial No. 10/259,176, filed on September 27, 2002, entitled MECHANISM FOR UNIFORM ACCESS CONTROL IN A DATABASE SYSTEM, by Ravi Murthy, Eric Sedlar, Nipun Agarwal, Sam Idicula, and Nicolas Montoya;

[0009] U.S. Patent Application Serial No. 10/256,777, filed on September 27, 2002, entitled LOADABLE UNITS FOR LAZY MANIFESTATION OF XML DOCUMENTS by Syam Pannala, Eric Sedlar, Bhushan Khaladkar, Ravi Murthy, Sivasankaran Chandrasekar, and Nipun Agarwal;

[0010] U.S. Patent Application Serial No. 10/260,381, filed on September 27, 2002, entitled MECHANISM TO EFFICIENTLY INDEX STRUCTURED DATA THAT PROVIDES HIERARCHICAL ACCESS IN A RELATIONAL DATABASE SYSTEM, by Neema Jalali, Eric Sedlar, Nipun Agarwal, and Ravi Murthy;

[0011] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled DIRECT LOADING OF OPAQUE TYPES, by Namit Jain, Ellen Batbouta, Ravi Murthy, Nipun Agarwal, Paul Reilly, and James Stenoish (Attorney Docket No. 50277-2236);

[0012] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled IN-PLACE EVOLUTION OF XML SCHEMAS, by Sam Idicula, Nipun Agarwal, Ravi Murthy, and Sivasankaran Chandrasekar (Attorney Docket No. 50277-2237); and

[0013] U.S. Patent Application Serial No. _____, filed on the same day herewith, entitled MECHANISM TO ENABLE EVOLVING XML SCHEMA, by Sam

Idicula, Nipun Agarwal, Ravi Murthy, Eric Sedlar, and Sivasankaran Chandrasekar (Attorney Docket No. 50277-2238).

FIELD OF THE INVENTION

[0014] The present invention relates to database systems, and in particular, to techniques for directly loading semistructured data into a database.

BACKGROUND OF THE INVENTION

[0015] Structured data conforms to a type definition. For example, a type definition for a "person" type may define distinct attributes such as "name," "birthdate," "height," "weight," and "gender." Each "instance" of a particular type comprises a separate value for each of the attributes defined by the particular type. For example, an instance of the "person" type might comprise values such as "Fred Brown," "January 1, 1980," "72 inches," "240 pounds," and "male." Each attribute is also of a type. For example, the "name" attribute might be of a "string" type, the "birthdate" attribute might be of "date" type, and the "gender" attribute might be of an "enumerated" type. Structured data might comprise multiple different instances of the same type.

[0016] Different approaches may be used to store structured data into a database. One such approach is called "conventional path loading." According to conventional path loading, a client application parses structured data that comprises one or more instances of a type. Values within the structured data correspond to attributes of the type. The client application generates Structured Query Language (SQL) commands, such as INSERT commands, that, when executed by a database server, cause the database server to insert the values into corresponding columns of a database table. Unfortunately, due to its heavy use of

the SQL engine, conventional path loading often suffers in terms of performance and memory consumption.

[0017] Another approach for storing structured data into a database is called "direct path loading." Through direct path loading, values within structured data are stored directly into a database without causing the SQL engine to load each row of data. By consulting a control file that is associated with the structured data, a client application can determine the types to which instances within the structured data conform. If the structures of the types are defined to the client application, then, based on those structures, the client application can create an array that comprises columns that correspond to the types' attributes. The client application can populate each attribute's corresponding column with values that correspond to that attribute. Once the array is populated, the client application can convert the array into a stream of data that conforms to the format of a database's data blocks. The client application then can stream the data to a database server, which can write the data directly into one or more data blocks in the database. Direct path loading exhibits performance superior to that of conventional path loading.

[0018] Some types indicated by a control file may be standard types that are defined to a client application. A scalar type is an example of such a standard type. The client application has information about the characteristics of a scalar type, such as the maximum storage size of a scalar type. With this information, the client can generate the data stream as described above.

[0019] However, some types indicated by a control file might not be among the types that are defined to the client application. A type indicated by a control file might have a structure that is defined only to a program that implements that type. Although the type

might comprise attributes that are of standard types, the control file and the client application might lack any information about the number or types of such attributes.

[0020] Without such information, the client application cannot generate or populate an array that comprises a separate column for each such attribute. The client application does not possess sufficient information to map values that correspond to such attributes to corresponding columns of a table in a relational database. Consequently, there is no effective way for the client application to store instances of such a type in a database using the direct path loading approach.

[0021] Types that are not defined to a client application are called "opaque types" relative to the client application, because the internal structure of such types is obscured from the client application. The internal structure of an opaque type, including the number and types of attributes of the opaque type, often are defined only to a program that implements the opaque type. Such a program may be external to both the client application and the database server.

[0022] It may not be practical to modify a client application every time that a new type is introduced, so that the new type is defined to the client application. Additionally, the structures of some existing types may change as time passes. It may be impractical to modify a client application every time that the structure of an existing type changes.

[0023] One kind of opaque type is an XML type. An example of an XML type is provided in co-pending U.S. Patent Application No. 10/259,278. "XML" stands for "Extensible Markup Language." An XML schema is metadata that describes a hierarchical structure. Instances of the XML schema comprise data that conforms to the structure described by the XML schema. Through XML elements expressed in the structure, an XML schema defines one or more types. XML elements in such a structure may be mapped to

columns of database tables. Using the conventional path loading approach, values that correspond to the XML elements may be stored in the columns that are mapped to those XML elements.

[0024] An XML type is special because an XML type may define alternative structures to which instances of the XML type may conform. For example, an XML type definition might indicate that one or more attributes of the XML type are optional. Therefore, if attributes "A," "B," and "C" are optional, then one instance of the XML type might comprise a value for attribute "A," but no values for attributes "B" or "C," while another instance of the XML type might comprise a value for attribute "B," but no values for attributes "A" or "C." Because the instances may conform to alternative defined structures rather than a single defined structure, the instances may be said to comprise "semistructured" data rather than "structured" data.

[0025] At present, client applications are unable to use the direct path loading approach effectively to store semistructured data. Because the direct path loading approach exhibits performance superior to that of the conventional path loading approach, a technique that overcomes the limitations of prior approaches to is needed.

[0026] The approaches described in this section are approaches that could be pursued, but not necessarily approaches that have been previously conceived or pursued. Therefore, unless otherwise indicated, it should not be assumed that any of the approaches described in this section qualify as prior art merely by virtue of their inclusion in this section.

BRIEF DESCRIPTION OF THE DRAWINGS

[0027] The present invention is illustrated by way of example, and not by way of limitation, in the figures of the accompanying drawings and in which like reference numerals refer to similar elements and in which:

[0028] Figure 1 is a block diagram that illustrates example structures that contain columns for storing values that are not to be displayed to a user;

[0029] Figure 2 is a block diagram that illustrates a row of a database table in which multiple values associated with an instance of a child type are stored in-line with values associated with an instance of a parent type;

[0030] Figure 3 is a block diagram that illustrates a row of a database table in which multiple values associated with an instance of a child type are stored out-of-line from a database table that stores values associated with an instance of a parent type;

[0031] Figure 4A is a block diagram that illustrates multiple rows of a nested database table in which multiple values associated with an instance of a child type are stored;

[0032] Figure 4B is a block diagram that illustrates multiple rows of a nested database table in which multiple values associated with an instance of a child type are stored out-of-line;

[0033] Figure 5 is a block diagram that illustrates a system, according to an embodiment of the present invention, in which semistructured data may be stored in a database according to the direct path loading approach;

[0034] Figures 6A-6D are flow diagrams that illustrate a technique, according to an embodiment of the present invention, for storing semistructured data in a database according to the direct path loading approach; and

[0035] Figure 7 is a block diagram that illustrates a computer system upon which an embodiment of the invention may be implemented.

DETAILED DESCRIPTION OF THE INVENTION

[0036] Techniques and systems are provided for storing semistructured data according to the direct path loading approach. In the following description, for the purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, that the present invention may be practiced without these specific details. In other instances, well-known structures and devices are shown in block diagram form in order to avoid unnecessarily obscuring the present invention.

FUNCTIONAL OVERVIEW

[0037] It is desirable to store semistructured data in a database according to the direct path loading approach. According to one embodiment of the present invention, a program that implements a type to which semistructured data conforms (a "type implementor") registers, with a client application, one or more routines that are associated with the type. The type implementor, which is external to both the client application and a database server that manages the database, implements the routines. In response to the registration, the client application adds an entry to a dispatch table. The entry indicates the association between the type and the routines.

[0038] The client application reads semistructured data that comprises instances of the type. By consulting a control file that is associated with the data, the client application determines that the instances are of the type. The structure of the type, the number of attributes of the type, and the types of those attributes are not defined to the client application. Therefore, the client application locates an entry in the dispatch table that indicates the specified type. The entry indicates the association between the specified type

and the routines that are implemented by the type implementor. For example, an entry might indicate an association between an XML type and a set of routines that are implemented by the XML type's implementor.

[0039] The client application invokes the routines. One or more of the routines creates an array for storing instances of the type. The array comprises a separate column for each attribute of the type. One or more of the routines populates the columns with values that are specified in the data. Each such value corresponds to a separate attribute of the type. One or more of the routines stores each such value in a column that corresponds to an attribute that corresponds to that value.

[0040] The array further may comprise one or more additional columns that correspond to one or more hidden columns of a database table that is to store at least parts of instances of the type. Hidden columns store values that are not displayed to a user when the database table that contains the hidden columns is queried.

[0041] One or more of the routines populates the additional columns with additional values that typically are not specified in the data. For example, for each row of the array, a routine might generate a different instance identifier to distinguish the instance stored in that row from other instances stored in other rows. The routine might populate one of the additional columns with the instance identifiers.

[0042] For another example, for each row of the array, a routine might generate a different positional descriptor that indicates the position of the various attributes within the instance stored in that row. Such positional descriptors are especially useful in relation to instances of an XML type. Although XML documents express data according to a structure in which each attribute value is located at a different position relative to other attribute values, database table rows that store the values of such XML instances typically are not

ordered in a way that indicates a position. By populating an additional column with positional descriptors, the original structure of an XML document can be preserved.

[0043] Attributes of one type may conform to other types. For example, a top-level XML type might indicate a first attribute that is of a scalar type, and a second attribute that is of a "purchase order" type. The "purchase order" type also might indicate several attributes. When a type is nested within another type in this manner, routines associated with the nested type are invoked according to the technique described above. For each nested type, the routines associated with that nested type create and populate a separate array that corresponds to that nested type. References to an array generated for a nested type are stored in a corresponding column of an array generated for the type that indicates an attribute that is of the nested type.

[0044] With the arrays populated, the client application converts the arrays, including any additional columns, into a data stream that conforms to the format of the database's data blocks. The client application then streams the data to a database server, which writes the data directly into one or more data blocks in the database. As a result, values that were stored in array columns that correspond to database table hidden columns are stored in those hidden columns.

[0045] Thus, the client application stores semistructured data in the database according to the direct path loading approach. Because the direct path loading approach does not require the SQL engine to load each row of data, the direct path loading approach is faster and consumes less memory than the conventional path loading approach described above. Because type implementors are external to both the client application and the database server, instances of new types can be stored in the database without modifying either the client application or the database server.

[0046] The code that implements the routines that are associated with a particular type may be centralized within the particular type's implementor rather than being distributed among multiple separate programmatic components. Such centralization promotes savings in terms of time and money.

ARRAY COLUMNS CORRESPONDING TO HIDDEN COLUMNS

[0047] As is described above, one or more routines associated with a particular type generate an array that contains separate columns for separate attributes of the particular type. The array may also contain one or more columns that correspond to hidden columns of a database table in which instances of the particular type are to be stored.

[0048] Figure 1 is a block diagram that illustrates example structures 100 that contain columns for storing values that are not to be displayed to a user. Structures 100 comprise a database table 102 and an array 104. Database table 102 comprises user-visible columns 106A-106N and hidden columns 108A-108N. User-visible columns 106A-106N correspond to attributes of a type whose instances are to be stored in database table 102.

[0049] Routines associated with a type whose instances are to be stored in database table 102 generate array 104. Array 104 comprises array columns 110A-110N and columns 112A-112N. Columns 110A-110N correspond to user-visible columns 106A-106N. Columns 112A-112N correspond to hidden columns 108A-108N.

[0050] The routines populate columns 110A-110N with values specified in semistructured data. The routines populate columns 112A-112N with additional values, which might not be specified in the semistructured data. For example, the routines may populate column 112A with instance identifiers that distinguish the instances stored in the

rows of array 104, and the routines may populate column 112B with positional descriptors that indicate positions of such instances relative to each other within an original document.

[0051] When the routines generate data streams based on array 104, they do so based on the structure of columns 110A-110N and columns 112A-112N and all of the values stored therein. Therefore, when the data are written to a database, values that were stored in columns 110A-110N are stored in user-visible columns 106A-106N, and values that were stored in columns 112A-112N are stored in hidden columns 108A-108N.

[0052] Hidden columns are useful for storing information that is needed for re-creating an original document from which values were obtained, especially when that information cannot be derived from the values themselves. For example, such information may indicate some structural aspects of an original document. Because such information is typically used by a computer program and not by a human user, hidden columns are especially appropriate for storing such information.

[0053] While an embodiment of the invention is described with reference to arrays, alternative embodiments may use data structures other than arrays to perform the techniques described herein.

OUT-OF-LINE TABLES AND NESTED TABLES

[0054] As is described above, a type may indicate multiple attributes, one or more of which may be of another type that indicates other attributes. A type of an attribute indicated by another type may be called a nested type. For example, if a type "A" indicates an attribute "X" that is of type "B," then "B" is called a nested type. The nested type and the type that indicates the attribute that is of the nested type may be called the child type and the parent

type, respectively, relative to each other. In the above example, type "A" is a parent type relative to type "B," and type "B" is a child type relative to type "A."

[0055] Values within instances that conform to nested types may be stored in database tables in any of a variety of ways. Because values may be stored in database tables in a variety of ways, techniques for directly loading semistructured data into database tables should accommodate each of those ways.

[0056] For example, given a database table that contains multiple columns for multiple attributes of a parent type, multiple values corresponding to multiple attributes of a child type may be stored together within a single column of that database table as a large object (LOB). Internal delimiters within the LOB may delimit the values that correspond to the child type. When values corresponding to a child type are stored in this manner, the values are said to be stored "in-line."

[0057] Figure 2 is a block diagram that illustrates a row of a database table 200 in which multiple values associated with an instance of a child type are stored in-line with values associated with an instance of a parent type. Values "A," "C," and "D," associated with the parent type, are stored in columns 202A, 202B, and 202D, respectively. Multiple values "(W,X,Y,Z)," associated with the child type, are stored in-line in column 202B as a LOB.

[0058] Alternatively, multiple values corresponding to multiple attributes of a child type may be stored in a database table that is separate from the database table that contains columns that correspond to attributes of the parent type. A column within the parent type's database table may store references to the child type's database table. Within the child type's database table, the values corresponding to the child type still may be stored in a single column as a LOB. When values corresponding to a child type are stored in this manner, the values are said to be stored "out-of-line."

[0059] Figure 3 is a block diagram that illustrates a row of a database table 304 in which multiple values associated with an instance of a child type are stored out-of-line from a database table 300 that stores values associated with an instance of a parent type. Values "A," "C," and "D," associated with the parent type, are stored in columns 302A, 302B, and 302D, respectively. Column 302B stores a reference to column 306 of database table 304. Multiple values "(W,X,Y,Z)," associated with the child type, are stored out-of-line in column 306 as a LOB.

[0060] However, multiple values do not need to be stored as a LOB. Each of the multiple values may be stored in a separate row of a single column in the child type's database table. A first row of the column may store a first of the values, a second row of the column may store a second of the values, and so on. Sets of values within separate instances may be stored in separate sets of rows in the same column of the child type's database table. For example, values of a first instance of the child type may be stored in the first N rows of the column, values of a second instance of the child type may be stored in the next N rows of the column, and so on. When values corresponding to a child type are stored in this manner, the values are said to be stored in a "nested table."

[0061] Figure 4A is a block diagram that illustrates multiple rows of a nested database table 404 in which multiple values associated with an instance of a child type are stored . Values "A," "C," and "D," associated with the parent type, are stored in columns 402A, 402B, and 402D, respectively. Column 402B stores a reference to rows 406A-408D of database table 404. Values "W," "X," "Y," and "Z," associated with the child type, are stored in rows 406A-408D, respectively, of nested table 404. In this case, the values are stored in separate rows, and not as a LOB.

[0062] Figure 4B is a block diagram that illustrates multiple rows of a nested database table 454 in which multiple values associated with an instance of a child type are stored out-of-line. Values "A," "C," and "D," associated with the parent type, are stored in columns 452A, 452B, and 452D, respectively. Column 402B stores a reference to a table of pointers 458. Each row in table of pointers 458 contains a pointer that points to a separate row of database table 454. Values "W," "X," "Y," and "Z," associated with the child type, are stored in rows 456A-458D, respectively, of nested table 454. In this case also, the values are stored in separate rows, and not as a LOB.

[0063] To associate the multiple rows in the child type's database table with the single row in the parent type's database table, the corresponding rows from both database tables may be associated with a set identifier that distinguishes those associated rows from other associated rows in the database tables. For example, the first row in the parent type's database table and the first N rows in the child type's database table may be associated with a first unique set identifier, the second row in the parent type's database table and the second N rows in the child type's database table may be associated with a second unique set identifier, and so on.

[0064] There is no limitation on the levels of indirection that may be used when storing instances of nested types in nested tables. A nested type may indicate an attribute that is of another nested type. Thus, a row of a nested table may indicate a reference to another out-of-line database table or nested table.

[0065] Based on characteristics of the nested type, one way of storing instances of the nested type may be selected over other ways. A schema processor may make such a selection.

[0066] A nested table notably is useful for storing a collection that may be defined by an XML schema. Such a collection comprises zero or more instances of a type. The XML schema may indicate a minimum and/or maximum number of instances of the type that are to be contained in the collection. A nested table corresponding to such a collection may be created to comprise a number of rows equal to the maximum number of instances that will be in the collection.

EXAMPLE SYSTEM FOR DIRECTLY LOADING SEMISTRUCTURED DATA

[0067] Figure 5 is a block diagram that illustrates a system 500 in which semistructured data may be stored in a database according to the direct path loading approach, according to an embodiment of the present invention. System 500 comprises a client application 502, a database server 504, a database 506, and type implementors 508A-N. Client application 502, database server 504, and type implementors 508A-N are coupled communicatively to each other. Database server 504 is coupled communicatively to database 506.

[0068] Client application 502 reads or otherwise receives semistructured data 518 as input. Semistructured data 518 comprises instances of a type. For example, semistructured data 518 may comprise one or more XML instances that conform to an XML schema. Semistructured data 518 also comprises an identity of the type. For example, the type may be identified as an XML type. Semistructured data 518 does not indicate the structure of the type. The structure of the type is not defined to client application 502.

[0069] Based on semistructured data 518, client application 502 determines the identity of the type to which the instances conform. Client application consults dispatch table 512 to find, within the dispatch table, an entry that indicates the identity of the type. Dispatch table 512 comprises a separate entry for each of type implementors 508A-508N. Each entry

indicates memory addresses of routines that are implemented by the type implementor that implements the type that is indicated by that entry. Table 1 below depicts an example of entries within a dispatch table.

TABLE 1—EXAMPLE DISPATCH TABLE ENTRIES

TYPE IDENTITY	ADDRESSES OF ROUTINES
Identity of type implemented by type implementor 508A	Address of routine 510AA . . . Address of routine 510AN
. . .	. . .
Identity of type implemented by type implementor 508N	Address of routine 510NA . . . Address of routine 510NN

[0070] The entries in dispatch table are added by client application 502 in response to type implementors 508A-N registering routines 510A-510NN with the client application. Each type implementor provides client application 502 with the information needed to add an entry for the type implemented by that type implementor. For example, type implementor

508A may load routines 510AA-AN into locations in memory, and then specify those locations to client application 502.

[0071] In response to finding, in dispatch table 512, an entry that indicates the identity of the type, client application 502 invokes the routines located at the memory addresses indicated by the entry. For example, if the type is implemented by type implementor 508A, then client application 502 invokes routines 510AA-510AN. If the type is an XML type, then client application 502 invokes routines that are associated with the XML type in the dispatch table.

[0072] One or more of the invoked routines creates an array 516A in client application address space 514. Client application address space 514 comprises a segment of memory allocated for use by client application 502. Array 516A comprises a separate column for each attribute of the type.

[0073] For example, an invoked routine may initialize a context for the type. Another invoked routine may indicate a number of columns in a database table that is to store instances of the type. For example, one implementation of an XML type maps to two top-level columns of a database table, in which one of the two top-level columns is a hidden column. The hidden column is for storing a positional descriptor, and the user-visible column is for storing a LOB or a reference to an out-of-line or nested table. Therefore, an invoked routine of the XML type implementor creates an array with two columns that correspond to the two top-level columns of the database table.

[0074] One or more of the invoked routines populates the columns of array 516A, including columns that correspond to hidden columns of a database table, with values. Some values may be specified in semistructured data 518, and other values may be derived from

values specified in the semistructured data. Each value specified in semistructured data 518 corresponds to a separate attribute of the type.

[0075] Client application 502 may pass such values as parameters to one or more of the invoked routines. For example, for each instance specified within semistructured data 518, client application 502 may pass a single block of combined values, which represent that instance, as a parameter to one or more routines. The one or more routines may parse the block of combined values to produce separate values that correspond to the separate attributes of the type.

[0076] One or more of the invoked routines stores the values in corresponding columns of array 516A. Each row of array 516A stores values for a different instance specified in semistructured data 518. One of more the invoked routines returns, to client application 502, one or more pointers to one or more addresses within client application address space 514 at which one or more populated rows of array 516A can be found. Using the one or more pointers, client application 502 can locate and read the populated rows of array 516A.

[0077] One or more of the attributes of the type may be of a nested type that indicates one or more other attributes. For example, a parent type might indicate a first attribute that is of a scalar type, and a second attribute that is of a "purchase order" child type. The "purchase order" type also might indicate several attributes. In this case, the routines associated with the parent type invoke routines associated with the child type. The client application does not need to be aware of or invoke routines associated with the child type. When a routine is invoked to describe the structure of the parent type, that routine invokes another routine to describe the structure of the child type. When a routine is invoked to create an array for the parent type, that routine invokes another routine to create an array for

the child type. When a routine is invoked to populate the array for the parent type, that routine invokes another routine to populate the array for the child type.

[0078] The number of columns in an array generated by a child type's implementor is based on the number of columns in a database table that is to store instances of the child type. If instances of a child type are stored in a nested table, then the array generated for the child type may contain one column that corresponds to the one column of the nested table, and each instance of the child type may be stored in a different row of the array column. Thus, multiple rows of a child array may correspond to a single row of a parent array. To preserve this correspondence, a set identifier is generated. The set identifier links the row of the parent array with the corresponding rows of the child array. The set identifier is stored in the parent array.

[0079] Each array may be populated independently of each other array. Arrays may be loaded and streamed asynchronously and independently.

[0080] Based on the populated rows of one or more arrays, the client application 502 generates a data stream. A data stream for one array may be generated independently of a data stream for another array. The data stream conforms to the format of data blocks within database 506. As a result, the data stream generated by client application 502 may be written directly to database 506 without causing the SQL engine to load each row of data. Client application 502 streams the data to database server 504. A stream generated based on one array may be sent to the database server independently of a stream generated based on another array. Database server 504 writes the data received from client application 502 directly into one or more data blocks in database 506. Values in array columns that correspond to hidden columns in database tables are stored in the corresponding hidden columns as a result of the writing.

MEMORY MANAGEMENT

[0081] Because the amount of memory available in client application address space 514 is limited, array 516A might not comprise enough rows to store, concurrently, values of all of the instances that are specified in semistructured data 518. Therefore, after a specified number of rows of array 516A have been populated, a data stream may be generated based on those populated rows, and the data may be streamed to database server 504.Then, the memory that the populated rows occupy may be freed. Once the memory has been freed, array 516A may be re-populated with values of additional instances for which a data stream has not yet been generated. This process may be repeated until data streams for all of the instances indicated in semistructured data 518 have been generated and streamed to database server 504.

[0082] Different arrays may be populated independently of each other. Different arrays may be associated with different database tables that comprise different numbers of rows. Values stored in such different arrays may be flushed to persistent storage at different times. Memory that stores one array may be freed before other memory that stores another array. For example, a parent array may be associated with a database table that comprises five rows, while a child array may be associated with a nested database table that comprises twenty-five rows. Rows in the child array may be populated, streamed to the database server, and re-populated multiple times before rows in the parent array are streamed to the database server once.

MANAGING NESTED TYPES

[0083] Semistructured data 518 might comprise instances of a type "A" that comprises two attributes: an attribute "B" of a scalar type and an attribute "C" of a type "D" that is implemented by type implementor 508A. In turn, type "D" might comprise two attributes "E" and "F," both of scalar types.

[0084] In this example, the structure of type "A" is defined, at a high level, to client application 502. Client application 502 possesses sufficient information to generate, in client application address space 514, an array (array "A") to store instances of type "A". Array "A" comprises a column (column "B") for attribute "B" and a column (column "C") for attribute "C."

[0085] Client application 502 populates rows of array "A" on a per-instance basis. Because attribute "B" is of a scalar type, client application 502 may populate column "B" without invoking any external routines. Because attribute "C" is of a type implemented by type implementor 508A, client application 502 invokes one or more of routines 510AA-AN for each instance of type "A." For each instance, client application 502 passes a combined value block, which represents the value of attribute "C" for that instance, to the routines.

[0086] For each instance, the routines generate, in client application address space 514, columns (columns "E" and "F") corresponding to attributes "E" and "F" of type "D." For each instance, the routines populate columns "E" and "F" with corresponding values of attributes "E" and "F" separated out from the combined value block received as a parameter. For each instance, the routines return, to client application 502, a pointer to populated columns "E" and "F". Client application 502 stores the pointer in column "C" in the instance's corresponding row of array "A."

[0087] Client application 502 does not possess sufficient information about type "D" to free memory that stores instances of type "D." Therefore, when client application 502 is going to free memory that stores values for attribute "B," the client application also invokes one or more of routines 510AA-AN to free memory that stores the corresponding values for attributes "E" and "F."

[0088] When client application 502 is going to free memory that contains a pointer to columns "E" and "F", the client application passes the pointer as a parameter to one or more of routines 510AA-AN. Those routines then free the memory to which the pointer points. Client application 502 does not free the memory that stores the pointer until the client application invokes the routines that free the memory to which the pointer points.

REFERENCE COUNTS

[0089] Multiple arrays may reference the same temporary data structure in memory. Such a data structure should be maintained in memory as long as at least one array is referencing the data structure. Only after no arrays are referencing a data structure should the memory that stores that data structure be freed for other purposes. Therefore, a separate reference count may be associated with each such temporary data structure in memory. When an array begins to make reference to a given data structure, then the reference count associated with that data structure is incremented. When an array no longer needs to make reference to a given data structure, then the reference count associated with that data structure is decremented. When a data structure's reference count is zero, then the memory that stores that data structure may be freed for other purposes.

EXAMPLE ROUTINES IMPLEMENTED BY A TYPE IMPLEMENTOR

[0090] Client application 502 may invoke different routines to perform different functions relative to a type. For example, each of routines 510AA-AN may perform a different function relative to a type implemented by type implementor 508. Such functions may include:

- Allocating and initializing a general context block;
- Indicating a type of a database table in which instances of the type are to be stored;
- Indicating a number of columns in a database table in which instances of the type are to be stored;
- Indicating types of columns in a database table in which instances of the type are to be stored;
- Initializing a context for a database table in which instances of the type are to be stored;
- Allocating memory, in client application address space 514, for one or more arrays to store values of instances of the type;
- Parsing combined value blocks and storing separated values into separate columns of one or more arrays;
- Freeing memory that stores values of an instance of a type;
- Flushing existing populated memory structures to persistent storage;
- Completing the direct path loading and freeing all memory that was allocated to perform the direct path loading; and
- Aborting the direct path loading and freeing all memory that was allocated to perform the direct path loading.

[0091] Different routines may accept different parameters and return different results. Client application 502 may invoke one or more of routines 510AA-510AN sequentially in an order designed to achieve the ultimate goal of storing instances of the type in database 506 according to the direct path loading approach.

ERROR MANAGEMENT

[0092] Client application 502 may specify actions to be performed when an error occurs during the performance of any of the techniques described herein. For example, when such an error occurs, client application 502 may update an error counter value and determine whether the error counter value is greater than a specified threshold. Client application 502 may indicate that the techniques currently being performed should continue, despite the errors, unless the error counter value is greater than the specified threshold. Client application 502 may indicate that the techniques currently being performed, and techniques that will be performed thereafter, should be aborted if the error counter value is greater than the specified threshold.

[0093] It is desirable for routines 510AA-NN to handle errors in a way that is consistent with the way that client application 502 handles errors. Therefore, according to one embodiment, client application 502 passes memory addresses of error handling routines, which are implemented by the client application, to routines 510AA-NN. When any of routines 510AA-NN determines that an error has occurred, that routine executes an error handling routine that is located at the specified memory address. Thus, if an error handling condition is satisfied during the execution of any of routines 510AA-NN, actions connected to the satisfaction of the condition will be performed just as if the condition had been satisfied outside of the execution of such a routine.

EXAMPLE TECHNIQUE FOR STORING SEMISTRUCTURED DATA IN A DATABASE ACCORDING TO THE DIRECT PATH LOADING APPROACH

[0094] Figures 6A-6D are flow diagrams that illustrate a technique 600, according to an embodiment of the present invention, for storing semistructured data in a database according to the direct path loading approach. Such semistructured data may comprise multiple different instances that conform to an XML schema.

[0095] In block 602, a "first" type implementor registers, with a client application, routines that are implemented by the first type implementor (the "first routines"). For example, type implementor 508A may register routines 510AA-AN with client application 502.

[0096] In block 604, the client application adds, to a dispatch table, an entry that indicates an association between the first routines and a parent type that the first type implementor implements. For example, client application 502 may add, to dispatch table 512, an entry that indicates an association between routines 510AA-AN and the type implemented by type implementor 508A. The parent type includes an attribute that is of a child type that is implemented by a "second" type implementor.

[0097] In block 606, the second type implementor registers, with the client application, routines that are implemented by the second type implementor (the "second routines"). For example, type implementor 508B may register routines 510BA-BN with client application 502.

[0098] In block 608, the client application adds, to the dispatch table, an entry that indicates an association between the second routines and the child type that the second type implementor implements. For example, client application 502 may add, to dispatch table

512, an entry that indicates an association between routines 510BA-BN and the type implemented by type implementor 508B.

[0099] In block 610, the client application receives semistructured data that specifies instances of the parent type. Each instance of the parent type specifies an instance of the child type. For example, client application 502 may read semistructured data 518, which may indicate (1) values of instances of the parent type and (2) the identity of the parent type.

[0100] In block 612, the client application determines, from the dispatch table, which routines are associated with the parent type. For example, client application 502 may determine, from dispatch table 512, that the parent type is associated with the first routines, routines 510AA-AN.

[0101] In block 614, the client invokes one or more of the first routines, which are associated with the parent type. For example, client application 502 may invoke routine 510AA to initialize a context block. Client application 502 may invoke routine 510AB to determine a type of a database table that will store instances of the parent type (the "parent database table"). Client application 502 may invoke routine 510AC to determine how many columns are in the parent database table. Client application 502 may invoke routine 510AD to determine the types of the columns in the parent database table.

[0102] In block 616, in response to its invocation, one or more of the first routines creates a "parent" array that comprises (1) a separate column for each attribute of the parent type and (2) columns that correspond to hidden columns of the parent database table. For example, when invoked, routine 510AE may create array 516A in client application address space 514. Array 516A might comprise two columns, one of which corresponds to a hidden column of a database table.

[0103] In block 622, one or more of the first routines invokes one or more of the second routines, which are associated with the child type. For example, routine 510AA may invoke routine 510BA to initialize a context block. Routine 510AB may invoke routine 510BB to determine a type of a database table that will store instances of the child type (the "child database table"). Routine 510AC may invoke routine 510BC to determine how many columns are in the child database table. Routine 510AD may invoke routine 510BD to determine the types of the columns in the child database table.

[0104] In block 624, in response to its invocation, one or more of the second routines creates a "child" array that comprises (1) a separate column for each attribute of the child type and (2) columns that correspond to hidden columns of the child database table. For example, when invoked, routine 510BE may create array 516B in client application address space 514. Array 516B might comprise five columns, two of which correspond to hidden columns of a database table.

[0105] Blocks 626-628 may be performed concurrently with blocks 630-632. In block 626, in response to its invocation, one or more of the first routines populates, with values of instances of the parent type, parent array columns that correspond to the attributes of the parent type. For example, when invoked, routine 510AF may populate one or more columns of array 516A with values of instances that are specified in semistructured data 518.

[0106] In block 628, in response to its invocation, one or more of the first routines populates, with other values, parent array columns that correspond to hidden columns of the parent database table. The other values might not be specified in the semistructured data, and might not be values that correspond to attributes of the parent type. However, the other values might be derived from the semistructured data. For example, when invoked, routine

510AF may populate some of the columns of array 516A with other values that are derived from semistructured data 518.

[0107] In block 630, in response to its invocation, one or more of the second routines populates, with values of instances of the child type, child array columns that correspond to the attributes of the child type. For example, when invoked, routine 510BF may populate one or more columns of array 516B with child type instance values that are specified in semistructured data 518.

[0108] In block 632, in response to its invocation, one or more of the second routines populates, with other values, child array columns that correspond to hidden columns of the child database table. The other values might not be specified in the semistructured data, and might not be values that correspond to attributes of the child type. However, the other values might be derived from the semistructured data. For example, when invoked, routine 510BF may populate some of the columns of array 516B with other values that are derived from semistructured data 518.

[0109] One or more set identifiers may be included among the values with which the parent and child arrays are populated. As described above, a set identifier associates a row in the parent array with a set of rows in a child array. Both the parent and child arrays may be generated with a column to store a set identifier.

[0110] In block 634, the client application generates a data stream based on the populated rows of one or more of the arrays. For example, based on populated arrays 516A and 516B, client application 502 may generate one or more data streams that conform to the format of data blocks stored in database 506.

[0111] In block 636, the client application streams the data to a database server 504. For example, client application 502 may stream data to database server 504.

[0112] In block 638, the database server writes the data directly into the database. For example, database server 504 may write data received from client application 502 directly into one or more data blocks in database 506. Values in array columns that correspond to hidden columns in database tables are stored in the corresponding hidden columns as a result of the writing.

[0113] Thus, semistructured data, such as instances of an XML schema, may be stored in a database according to the direct path loading approach. As is discussed above, the direct path loading approach is faster and consumes less memory than the conventional path loading approach. Using the techniques and systems described above, instances of an XML type may be stored in a database even after the XML type definition has been extended, without modifying either the client application or the database server.

HARDWARE OVERVIEW

[0114] Figure 7 is a block diagram that illustrates a computer system 700 upon which an embodiment of the invention may be implemented. Computer system 700 includes a bus 702 or other communication mechanism for communicating information, and a processor 704 coupled with bus 702 for processing information. Computer system 700 also includes a main memory 706, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 702 for storing information and instructions to be executed by processor 704. Main memory 706 also may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 704. Computer system 700 further includes a read only memory (ROM) 708 or other static storage device coupled to bus 702 for storing static information and instructions for processor 704. A

storage device 710, such as a magnetic disk or optical disk, is provided and coupled to bus 702 for storing information and instructions.

[0115] Computer system 700 may be coupled via bus 702 to a display 712, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 714, including alphanumeric and other keys, is coupled to bus 702 for communicating information and command selections to processor 704. Another type of user input device is cursor control 716, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 704 and for controlling cursor movement on display 712. This input device typically has two degrees of freedom in two axes, a first axis (e.g., x) and a second axis (e.g., y), that allows the device to specify positions in a plane.

[0116] The invention is related to the use of computer system 700 for implementing the techniques described herein. According to one embodiment of the invention, those techniques are performed by computer system 700 in response to processor 704 executing one or more sequences of one or more instructions contained in main memory 706. Such instructions may be read into main memory 706 from another computer-readable medium, such as storage device 710. Execution of the sequences of instructions contained in main memory 706 causes processor 704 to perform the process steps described herein. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions to implement the invention. Thus, embodiments of the invention are not limited to any specific combination of hardware circuitry and software.

[0117] The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 704 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and

transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 710. Volatile media includes dynamic memory, such as main memory 706. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 702. Transmission media can also take the form of acoustic or light waves, such as those generated during radio-wave and infra-red data communications.

[0118] Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic tape, or any other magnetic medium, a CD-ROM, any other optical medium, punchcards, papertape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

[0119] Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 704 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 700 can receive the data on the telephone line and use an infra-red transmitter to convert the data to an infra-red signal. An infra-red detector can receive the data carried in the infra-red signal and appropriate circuitry can place the data on bus 702. Bus 702 carries the data to main memory 706, from which processor 704 retrieves and executes the instructions. The instructions received by main memory 706 may optionally be stored on storage device 710 either before or after execution by processor 704.

[0120] Computer system 700 also includes a communication interface 718 coupled to bus 702. Communication interface 718 provides a two-way data communication coupling to a network link 720 that is connected to a local network 722. For example, communication

interface 718 may be an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. As another example, communication interface 718 may be a local area network (LAN) card to provide a data communication connection to a compatible LAN. Wireless links may also be implemented. In any such implementation, communication interface 718 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

[0121] Network link 720 typically provides data communication through one or more networks to other data devices. For example, network link 720 may provide a connection through local network 722 to a host computer 724 or to data equipment operated by an Internet Service Provider (ISP) 726. ISP 726 in turn provides data communication services through the world wide packet data communication network now commonly referred to as the "Internet" 728. Local network 722 and Internet 728 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 720 and through communication interface 718, which carry the digital data to and from computer system 700, are exemplary forms of carrier waves transporting the information.

[0122] Computer system 700 can send messages and receive data, including program code, through the network(s), network link 720 and communication interface 718. In the Internet example, a server 730 might transmit a requested code for an application program through Internet 728, ISP 726, local network 722 and communication interface 718.

[0123] The received code may be executed by processor 704 as it is received, and/or stored in storage device 710, or other non-volatile storage for later execution. In this manner, computer system 700 may obtain application code in the form of a carrier wave.

[0124] In the foregoing specification, embodiments of the invention have been described with reference to numerous specific details that may vary from implementation to implementation. Thus, the sole and exclusive indicator of what is the invention, and is intended by the applicants to be the invention, is the set of claims that issue from this application, in the specific form in which such claims issue, including any subsequent correction. Any definitions expressly set forth herein for terms contained in such claims shall govern the meaning of such terms as used in the claims. Hence, no limitation, element, property, feature, advantage or attribute that is not expressly recited in a claim should limit the scope of such claim in any way. The specification and drawings are, accordingly, to be regarded in an illustrative rather than a restrictive sense.